Universitat
Autònoma
de Barcelona

escola
d'enginyeria

2035

Implementació de les físiques d'un vehicle en un videojoc 3D

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per

Gaizka López Carrió

i dirigit per

Enric Martí Gòdia i
Jorge Arnal Montoya

Bellaterra, **21 de juny de 2010**.

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, **Enric Martí Gòdia**

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

Gaizka López Carrió

I per tal que consti firma la present.

Signat: **Enric Martí Gòdia**

Bellaterra, **21 de juny de 2010**.

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, **Jorge Arnal Montoya**

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

Gaizka López Carrió

I per tal que consti firma la present.

Signat: **Jorge Arnal Montoya**

Bellaterra, **21 de juny de 2010**.

"To finish first, you must first finish",
Rick Mears.

Prefacio

Corría el año 1983 en los relojes de entonces...

En la caja tonta, el americano Freddie Spencer lograba en el Gran Premio de San Marino su primer título de 500cc en la 35ª edición del Campeonato del Mundo de Motociclismo; en el sofá de casa, un renacuajo con apenas seis meses de vida daba la lata... ¡¿cómo entender que los fines de semana había alguien que hacía más ruido que él?!

27 años después, lejos queda el dominio de los pilotos norteamericanos en el mundial durante la década de los 80. No así mi pasión por el motociclismo, extendida también al resto de las modalidades del mundo del motor y entre las que podemos encontrar la virtual[1], culpable e impulsora de éste mi proyecto.

Ahora, la historia, siempre caprichosa ella, vuelve a situar en mi punto de mira las Américas, donde las compañías de videojuegos Papyrus (1987)[WKa] en los años 90 e iRacing[2] (2004)[WKb] en la actualidad (siempre con Dave Kaemmer[MGC] a la cabeza), han marcado la pauta en el mundo de los simuladores y las carreras *online*.

Con su trabajo como clara referencia y la intención de algún día relacionar mi afición con mi vida laboral, nace así pues este proyecto.

Bienvenidos a mi aventura.

[1] Ésta se denomina *sim racing*[WKc].
[2] Página web: http://www.iracing.com/

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La primera parte de este capítulo describe de manera breve y concisa las razones por las que he llevado a cabo este proyecto. La segunda parte inicia al lector en la temática del mismo y la problemática que ésta presenta, para finalmente exponerle la viabilidad del proyecto.

1.1. Organización de la memoria

Esta memoria contiene toda la documentación generada durante el desarrollo del proyecto, que no la del software generado: ésta última puede encontrarse en su correspondiente manual[1].

El texto se divide en 4 capítulos:

Capítulo 1 describe la razón de ser del proyecto e introduce al lector en el temario objeto de estudio y trabajo en los siguientes capítulos.

Capítulo 2 describe y cataloga los diferentes campos y facetas tratados durante el desarrollo del proyecto y recoge el trabajo realizado en cada uno de ellos.

Capítulo 3 expone los éxitos cosechados al final del camino así como los puntos flacos y/o ciertas incorrecciones.

Capítulo 4 contiene una valoración global de la realización del proyecto y enumera las mejoras y/o ampliaciones más relevantes de cara a un futuro desarrollo.

[1] *Manual de usuario* @ anexo A

1.2. Objetivo del proyecto

El objetivo de este proyecto es la construcción de una demo que implemente las físicas de un vehículo sobre un terreno 3D.

La idea principal es conocer y trabajar con los motores de física que se utilizan actualmente en los videojuegos, realizando en primera instancia un estudio de las diferentes alternativas existentes hoy día en el mercado, tanto de software propietario como libre, para luego seleccionar e implementar una de ellas.

La parte de implementación, la más representativa, consistirá en programar una capa que proporcione a un *framework*[2] el acceso a las siguientes funcionalidades: las colisiones estáticas y la física para controlar un coche sobre un terreno. Dicho *framework* será el utilizado en el curso de "Programación básica y avanzada de videojuegos 3D", materia que se imparte en ésta nuestra facultad, la ETSE.

1.3. Motivación

Ya de pequeño, mi padre y la televisión desarrollaron en mí una tremenda afición por los deportes de motor, motivo por el cual durante los últimos 15 años he vivido y participado de la evolución que ha experimentado el sector de la simulación (que no *arcade*[3]) de las disciplinas de motor en los videojuegos.

El deseo de querer algún día participar en dicha evolución y no tan sólo de ella es, pues, mi motivación para realizar este proyecto, el cual supone, tras cursar en el pasado "Gráficos por Computador II" y "Programación básica y avanzada de videojuegos 3D", el siguiente paso en el camino, más ahora que las físicas son la punta de lanza en el mundo de los videojuegos y están avanzando y evolucionando a pasos agigantados.

[2] Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser organizado y desarrollado[WKd].

[3] Dícese de los videojuegos clásicos o que recuerdan a las máquinas del mismo nombre. También se usa para diferenciar a los simuladores: en este sentido, suele referirse a los juegos relativamente fáciles de jugar o que no responden fielmente a la gravedad y otras fuerzas físicas reales, que era lo que predominaba como género en las máquinas[WKe].

1.4. Estado del arte

Desde los inicios, en pro de la experiencia final del usuario y/o la jugabilidad[4], siempre se ha tratado de lograr que, aquellos elementos de un videojuego contra los que interactúa el usuario, se comporten de manera mínimamente realista. Por ello, el concepto de la física no es algo nuevo en el mundo de los videojuegos.

Si bien al principio ésta era muy básica y reducía las leyes de la misma a la mínima expresión (constantes), con el tiempo, el aumento de la capacidad de cómputo permitió el uso de modelos físicos algo más complejos, aunque la realidad es que la física seguía desempeñando un papel secundario.

Lo que primaba entonces eran los gráficos, y por ello a finales de la década de los 90 aparecieron las tarjetas gráficas y revolucionaron el mercado. Ello, curiosa e indirectamente, fue el origen de la creación de los motores de física. Las tarjetas gráficas liberaron de una gran carga a la CPU, y ello lo aprovechó Havok para desarrollar una nueva tecnología en el mundo de los videojuegos: los motores de física.

Havok salió al mercado en el año 2000[WKg], y logró que los videojuegos alcanzaran una nueva dimensión en cuanto a realismo. AGEIA fue más allá y en el 2002 desarrolló una tarjeta dedicada única y exclusivamente a realizar los cálculos de su propio motor de físicas[WKh]. Durante los años siguientes, ambos motores fueron la referencia de esta tecnología, pero ésta tampoco tuvo la repercusión esperada a nivel comercial, quedando en un segundo plano. Sin embargo, a finales de 2007 y en tan sólo un año, todo cambió y los motores de física son ahora la gran apuesta de presente y futuro en el sector de los videojuegos: en setiembre de 2007 Intel adquirió Havok[INT], para luego Nvidia comprar AGEIA (ahora PhysX) en febrero de 2008[NVI] y posteriormente en junio del mismo año AMD cerrar un acuerdo con Havok (a pesar de Intel)[AMD].

Tras la nueva configuración de mercado y el enorme empuje que la comunidad está dando a Bullet (la gran alternativa de licencia zlib/libpng[5] nacida en 2005), Havok y Nvidia han cambiado su política y ahora ambos motores de física disponen de licencia gratuita para aplicaciones no comerciales. Veremos qué nos depara el futuro.

[4] Término empleado en el diseño y análisis de juegos que describe la calidad del juego en términos de sus reglas de funcionamiento y de su diseño como juego[WKf].

[5] Licencia de software libre[OSI].

1.5. Problemática

Para la realización de este proyecto, la problemática a resolver en primera instancia será la búsqueda y elección del motor de física más adecuado, ya que existe un amplio abanico de alternativas además de las mencionadas en el apartado anterior. Por ello, realizaremos un estudio comparativo entre ciertos motores, aunque sin extendernos demasiado, ya que será en la fase posterior cuando nos enfrentemos al bloque de la problemática principal: la explotación en sí del motor.

Dicho bloque constará de tres partes diferenciadas:

- hacer funcionar el motor a nivel global (inicialización y configuración).
- extraer la física que simule el comportamiento de un vehículo.
- comprender el funcionamiento del motor en sí para así poder establecer la correcta interrelación del motor de física con el *framework* del curso de "Programación básica y avanzada de videojuegos 3D", es decir, integrarlo en nuestro videojuego.

1.6. Trabajos relacionados

Debido a que los motores de física hicieron su aparición en el mercado casi 10 años atrás ya, no es difícil encontrar una gran variedad de títulos comerciales que hayan implementado la física de todo tipo de vehículos.

Las propias páginas web de Havok y PhysX, poseen un listado donde publicitan todos aquellos videojuegos que hacen uso (ya sea en parte o en su totalidad) de su motor[HAa][PHX]. Bullet enfoca el tema de modo completamente distinto y posee un subforo[BUL] dedicado al anuncio de toda aplicación, comercial o no, que implemente su motor.

En relación a este proyecto, destacaremos, por su similitud conceptual en cuanto a objetivos y el trabajo que muestra, una demo de un proyecto no comercial que encontramos precisamente en el subforo de Bullet:

- Bullet's Vehicle Simulation Test,
  http://code.google.com/p/game-ws/

1.7. Estudio de viabilidad

Siendo la base de nuestro proyecto el *framework* utilizado en el curso de "Programación básica y avanzada de videojuegos 3D", se ha procedido a revisar su lista de requisitos teniendo en cuenta las implicaciones derivadas de la integración de un motor de física.

Requisitos hardware

- PC con una vida no superior a los 4 años que soporte DirectX 9.0c.
- Si bien antes AGEIA sí requería de una tarjeta especial dedicada única y exclusivamente al cómputo de las físicas, ahora ya no es necesaria tras haber Nvidia incorporado dicha funcionalidad a sus tarjetas gráficas GeForce de generación 8 y superiores[KOW].

Requisitos software

- Entorno de desarrollo C++ Visual Studio 2005.
- DirectX SDK.
- Librerías de software libre OpenAL y LUA Scripting.
- Motores de física Havok, PhysX, Bullet & Oxford Dynamics.
- Generador de terrenos Terragen.

Disponibilidad hardware/software

- Las especificaciones del hardware del que disponemos tanto en la universidad como en casa cumplen suficientemente los requisitos mínimos.
- Todo el software listado está disponible ya sea en la universidad u online y con licencias válidas además de legales.

Coste

- Cabe destacar que, al no haber ningún requisito hardware especial y ser todas las licencias de coste gratuito, ya sea por ser estudiante de la UAB o por la característica no comercial de la aplicación, el proyecto es totalmente viable a nivel económico y no supondrá ningún coste extra al departamento.

Plazos de desarrollo y de entrega final

- La planificación del proyecto se ha realizado aplicando suficiente margen de seguridad en cada una de sus fases para asegurar así su viabilidad temporal.

1.7.1. Planificación

La tabla 1.1 muestra en detalle las diferentes tareas de las que consta cada fase del proyecto junto con sus respectivos tiempos estimados de realización.

Fases y tareas correspondientes	Fechas estimadas
Inicio del proyecto	*21/09/2009 - 31/10/2009*
- autodocumentación general	3 semanas
- estudio comparativo de motores de física	3 semanas
Explotación del motor de física	*01/11/2009 - 24/12/2009*
- funcionamiento y configuración	8 semanas
Integración de los módulos principales	*15/01/2010 - 01/03/2010*
- física del vehículo	3 semanas
- motor de terrenos	1 semana
- skybox	1 semana
- creación de niveles	1 semana
Pulir y completar lo anterior	*01/03/2010 - 01/05/2010*
- opcionales (IA, niveles, más funcionalidades)	6 semanas
- empezar a generar la documentación	2 semanas
Memoria del proyecto	*01/05/2010 - 22/06/2010*
- completar la redacción de la memoria	7 semanas
Preparar la lectura	*23/06/2010 - 30/06/2010*

Tabla 1.1: Planificación del proyecto

Capítulo 2

Desarrollo

La primera parte de este capítulo pone definitivamente al lector en situación para luego mostrarle el trabajo fruto del esfuerzo de todo el curso.

2.1. Marco del proyecto

2.1.1. ¿Qué es un motor de física?

Un motor de física es un software que proporciona la simulación de ciertos sistemas mecánicos simples, como por ejemplo la mecánica de sólidos rígidos[WKi] (incluyendo detección de colisiones), la mecánica de sólidos deformables[WKj] o la mecánica de fluidos[WKk], siendo la calidad de dicha simulación de alta precisión cuando ésta es computada por aplicaciones gráficas con fines científicos o de producción cinematográfica.

Sin embargo, en la actualidad, el mayor uso de los motores de física se da en la industria de los videojuegos, donde es más importante la velocidad de la simulación que no su grado de precisión. Ello es debido a que, el ya de por sí alto coste computacional que supone la simulación de las leyes de la física, ni debe ni puede afectar dramáticamente al *frame rate*[1] de un videojuego, circunstancia que sería inadmisible además de inviable para su jugabilidad.

Por ello, en esta especialidad se utilizan los denominados motores de física en tiempo real, que simplifican la complejidad de sus cálculos en pro de un aumento considerable de la velocidad de los mismos.

[1] Medida (en imágenes por segundo) de la frecuencia a la cual un reproductor de imágenes genera distintos fotogramas[WKl].

2.1.2. ¿Qué queremos obtener de él?

A nivel práctico, un motor de física es un programa que calcula cómo cierta clase de objetos deben comportarse en un escenario bajo la influencia de la mecánica newtoniana[WKm].

Lo que se desea, mediante su utilización, es mejorar la sensación de realismo que un jugador experimenta al interactuar con las diferentes entidades físicas situadas en la escena, haciéndole creer que éstas se comportan del mismo modo que lo harían en el mundo real. Así, un balón caerá de la mesa al ser golpeado, rebotará contra el suelo un número finito de veces y, si llega a un terreno con cierta pendiente negativa, bajará rodando por ella.

En el caso concreto de este proyecto, se busca obtener el comportamiento realista de un vehículo sobre un terreno 3D, siendo las claves a simular las reacciones del vehículo al manejo del mismo y las colisiones estáticas de éste contra el terreno y los distintos objetos que en él se encuentren.

2.2. Elección del motor de física

El desarrollo de un motor de física es algo verdaderamente complejo ya que requiere un conocimiento muy elevado de muchas y distintas áreas de la física. Por ello, la lista de motores de física existentes hoy en día[RHO] sigue creciendo, pues cada vez son más los pequeños proyectos *open source* que deciden especializarse únicamente en ciertos fenómenos físicos.

Elegir uno de ellos no fue tarea fácil: primero ya fue una sorpresa descubrir que existían bastantes más de los que esperábamos, y luego caímos en la inevitable trampa de preguntarnos: "¿cuál es el mejor de ellos?".

En nuestro afán por responder dicha pregunta, los pocos estudios comparativos[LJa][LJb] que encontramos estaban ya obsoletos u omitían motores importantes en su análisis, por lo que poco provecho pudimos extraer de ellos, pero finalmente encontramos un magnífico artículo[NOT] que nos devolvió al buen camino, haciéndonos comprender que existía una respuesta diferente para cada caso.

Efectivamente, la complejidad inherente al desarrollo de un motor de física facilita que cada uno de ellos, completo o no, tenga ciertas áreas más trabajadas y aptas que otros para la simulación de ciertos fenómenos físicos.

2.2.1. El grupo de los elegidos

En primera instancia, y con la ayuda de mi tutor, hice una primera criba del listado de motores existentes, seleccionando los siguientes: PhysX[2] y Havok[3] por ser ambos la referencia en el mercado, Bullet[4] como la emergente alternativa de código abierto y Oxford Dynamics[5] como único proyecto existente especializado en la física de vehículos.

Destacar que la mayoría de los motores no seleccionados (Tokamak, Newton, Open Dynamics, ...) fueron descartados por no incorporar una solución específica para la simulación de vehículos o por su práctica inexistencia entre los videojuegos actuales.

2.2.2. Requisitos a evaluar

Con el claro propósito de encontrar aquel motor con el que teóricamente sería más factible alcanzar los objetivos definidos en el apartado 1.2, establecimos una serie de parámetros que sirvieran de base para la evaluación y comparativa de los cuatro motores seleccionados:

Vehicle kit evalúa la existencia y las características de un módulo dedicado específicamente a la simulación física de vehículos: ¿cuán elevado es el grado de abstracción del módulo?, ¿cuán elevado es el conocimiento previo de física que se le exige al usuario para su uso y/o comprensión?

Documentación evalúa la calidad y cantidad de la misma, así como cuán bueno es el soporte a la comunidad de usuarios: ¿están bien documentadas las funciones del SDK?, ¿pueden encontrarse demos/ejemplos que muestren la implementación de ciertas funcionalidades?, ¿existen foros y/o listas de correo?

Repercusión evalúa cuán utilizado es entre aquellos videojuegos que incorporan la tecnología de los motores de física.

Situación tecnológica evalúa cuán reciente es la última versión del SDK: ¿está al día o, por el contrario, obsoleta?, ¿con qué frecuencia se publican nuevas actualizaciones?

[2]Página web: http://www.nvidia.com/object/physx_new.html
[3]Página web: http://www.havok.com/
[4]Página web: http://bulletphysics.org/
[5]Página web: http://oxforddynamics.com/index.html

2.2.3. Análisis

Las valoraciones efectuadas en éste y el siguiente apartado se basaron única y exclusivamente en mi juicio tras visitar la página web oficial de cada producto, descargar la versión más reciente de su software, ojear su correspondiente manual y ejecutar las diferentes demos incluidas en el paquete.

La puntuación otorgada se comprende entre 1 y 5, siendo esta última la más alta.

FastCar Library (Oxford Dynamics) @ 25 de noviembre de 2004
Su atractivo principal reside en ser la única del grupo desarrollada específicamente para simular las físicas de un vehículo. El gran temor, en consecuencia, era que su nivel de abstracción fuera prácticamente nulo (requiriendo un conocimiento de la física de la automoción del que no dispongo), pero la realidad es que éste es suficiente y viene acompañado de una documentación trabajada y de gran utilidad. Incorpora cinco demos, aunque tan sólo la de la figura 2.1 es de interés, y el comportamiento del vehículo es convincente pero no definitivo.

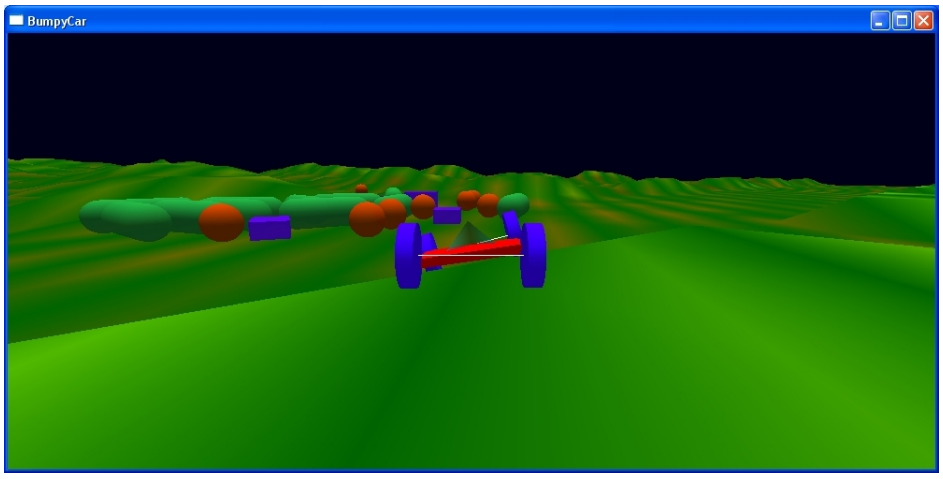


Figura 2.1: Trabajo de las suspensiones en la demo de Oxford Dynamics

Desafortunadamente no sólo no hay soporte a la comunidad, sino que el proyecto, cuya última actualización fue cinco años atrás, está abandonado y tan siquiera está operativa la lista de correo que entregaba las licencias gratuitas, básicas para el funcionamiento de la librería. Todo ello implica, obviamente, que su uso en los videojuegos de hoy en día sea nulo.

Vehicle kit	Documentación	Repercusión	Sit. tecnológica	Promedio
5	4	1	1	2.75

Tabla 2.1: Evaluación de Oxford Dynamics

Bullet 2.74 physics sdk @ 5 de marzo de 2009
Su atractivo principal reside en la gran comunidad abierta que lo soporta y desarrolla. Sin embargo, como la mayoría de los proyectos de tal naturaleza, peca de una documentación muy pobre. Así pues, y a pesar de disponer de un módulo especializado para la física de vehículos, éste está muy verde todavía y en la documentación sólo se le dedica un párrafo que te invita a mirar la demo y el código fuente de la misma, cuando la demo en sí es muy simplona. Por todo ello, es lógico que su presencia en el mercado sea limitada, pero si continúa con el alto nivel de evolución mostrado hasta ahora, se le augura un futuro prometedor.

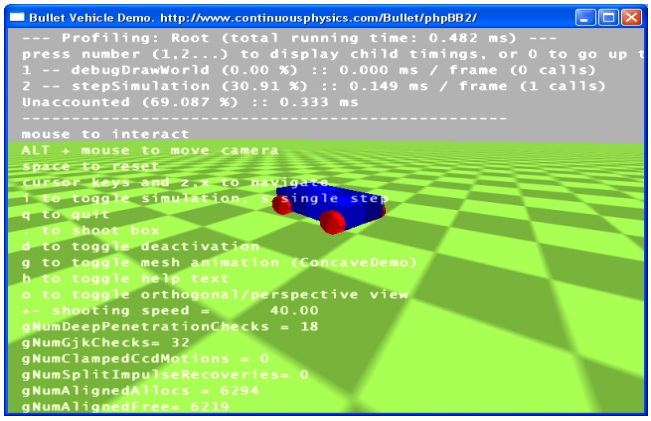


Figura 2.2: Demo de Bullet physics

Vehicle kit	Documentación	Repercusión	Sit. tecnológica	Promedio
2	1	3	5	2.75

Tabla 2.2: Evaluación de Bullet physics

Nvidia PhysX sdk 2.8.3 @ 30 de setiembre de 2009
Cabe reconocer que, desde un principio, era nuestro favorito: siendo la referencia en el mercado, más lo hablado con el profesor Enric Vergara mientras dábamos forma al proyecto en verano... todo apuntaba a que PhysX sería el elegido, y eso es difícil de olvidar por muy objetivo que luego uno quiera ser.

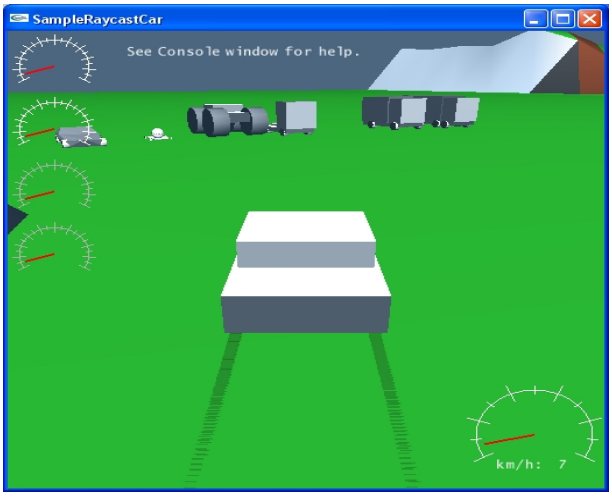


Figura 2.3: Parrilla de vehículos que incorpora la demo de PhysX

Curiosamente, cuando procedimos a su evaluación, a medida que más fuimos descubriendo más nos decepcionó. El soporte a la comunidad es correcto, aunque es sumamente complejo encontrar lo que uno desea (especialmente si también tienes que descubrir que es lo que deseas). El módulo específico para la física de vehículos sabe salir bien en la foto, pero en vivo no luce tanto. No ayuda una documentación sorprendentemente pobre, de unas cinco páginas, que básicamente se limita a describir la API y poco más. La demo sí es muy completa e incorpora una gran variedad de vehículos, pero el comportamiento de todos ellos no es nada atractivo. Por último, destacar que a nivel de desarrollo ha sabido reaccionar al ritmo endiablado de los proyectos de código abierto.

Vehicle kit	Documentación	Repercusión	Sit. tecnológica	Promedio
3	3	5	4	3.75

Tabla 2.3: Evaluación de Nvidia PhysX

Havok Physics 6.6.0 @ 4 de julio de 2009
La otra gran referencia en el mercado, no sólo por motivos históricos sino por su uso en videojuegos de gran renombre, era la que menos publicitaba su *vehicle kit* entre sus funcionalidades. Sin embargo, éste está bien trabajado, tiene un buen nivel de abstracción y, lo mejor de todo, una documentación que es todo un lujo: veinticinco páginas ni más ni menos, además de una API muy bien documentada. Todo ello acompañado de nueve demos que muestran todo tipo de vehículos (incluida una motocicleta) cuyo comportamiento es ciertamente convincente.

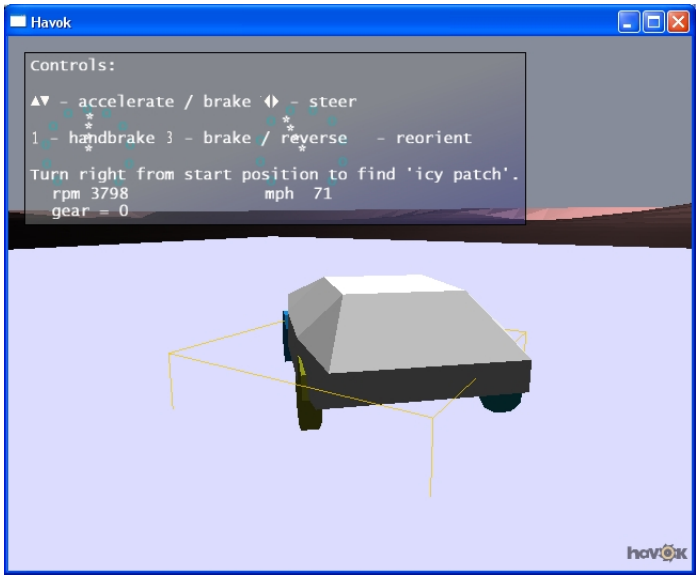


Figura 2.4: Vehículo deslizando sobre una capa de hielo en la demo de Havok

En cuanto al ritmo de desarrollo, también han sabido reaccionar a los nuevos tiempos y disponen de un foro dedicado al soporte de la comunidad en el que también participan miembros de Havok.

Vehicle kit	Documentación	Repercusión	Sit. tecnológica	Promedio
4	5	5	4	4.50

Tabla 2.4: Evaluación de Havok Physics

2.2.4. Comparativa

Para facilitar la comparativa, reagrupamos los resultados de todas las evaluaciones en la tabla 2.5:

Motor de física	Valoración
Oxford Dynamics	2.75
Bullet	2.75
PhysX	3.75
Havok	4.50

Tabla 2.5: Comparativa de los motores de física

Claramente Havok es el motor de física que más se ajusta a nuestras necesidades y, si tenemos en cuenta mi absoluto desconocimiento del uso de esta tecnología, la gran cantidad de ejemplos y documentación que incorpora adquiere un valor todavía mayor: la elección es Havok.

2.3. The name of the game

Una vez ya puesto al descubierto el protagonista del proyecto, este apartado describe cuáles han sido sus compañeros de viaje, es decir, los elementos de trabajo en nuestro día a día.

2.3.1. Fundamentos teóricos

Toda la teoría impartida en la asignaturas de "Gráficos por Computador II"[CAa] y en los cursos de "Programación básica y avanzada de videojuegos 3D"[CAb][CAc] ha sido no sólo la base de este proyecto, sino material de consulta habitual, siendo de especial utilidad toda la parte de geometría 3D y su correspondiente renderización[6].

[6]Término usado en jerga informática para referirse al proceso de generar una imagen desde un modelo[WKn].

2.3.2. Documentación

A nivel práctico, debido a mi completo desconocimiento de la tecnología Havok, sin duda fue primordial echar mano de su numerosa y extensa documentación:

- El manual de inicio rápido[HAb], una breve pero eficaz guía de iniciación a Havok Physics, muy útil para la correcta configuración de las librerías y el uso de la herramienta Visual Debugger.

- El manual de usuario[HAc], la biblia sin duda de este motor de física y en la que se describen absolutamente todos sus entresijos con sumo detalle.

- El manual de usuario de Havok Content Tools[HAd], vital para la correcta instalación y posterior uso de este *plugin* para Autodesk 3ds Max que nos permite exportar mallas 3D al mundo físico.

- El subforo dedicado de Havok que Intel soporta de manera oficial para la comunidad de usuarios[HAe], con un nivel de actividad muy respetable y lleno de valientes cuyas aventuras con el *vehicle kit* fueron de gran ayuda gracias a las respuestas que aportaron al foro los propios miembros de Havok.

Otras fuentes que fueron consultadas de manera más puntual pueden encontrarse debidamente referenciadas en la bibliografía[DTX][EMG][PHT][MAR].

2.3.3. Punto de partida

Hemos mencionado en numerosas ocasiones al *framework* del curso de "Programación básica y avanzada de videojuegos 3D" como la base de este proyecto, sin embargo, todavía no hemos explicado de qué trata. Pues bien, dicho *framework* es un videojuego completo que emula al famoso Quake III Arena[WKñ], un *FPS*[7] en el que desahogarse matando enemigos.

A primera vista, es difícil encontrarle alguna similitud con un videojuego de vehículos, sin embargo, no es ese el propósito que buscamos, sino la utilización de toda su estructura lógica de funcionamiento, ya que toda aplicación gráfica en tiempo real utiliza el mismo esqueleto.

[7] Del inglés *first-person shooter*, es un género de videojuegos de disparos que se desarrolla desde la perspectiva del personaje protagonista[WKo].

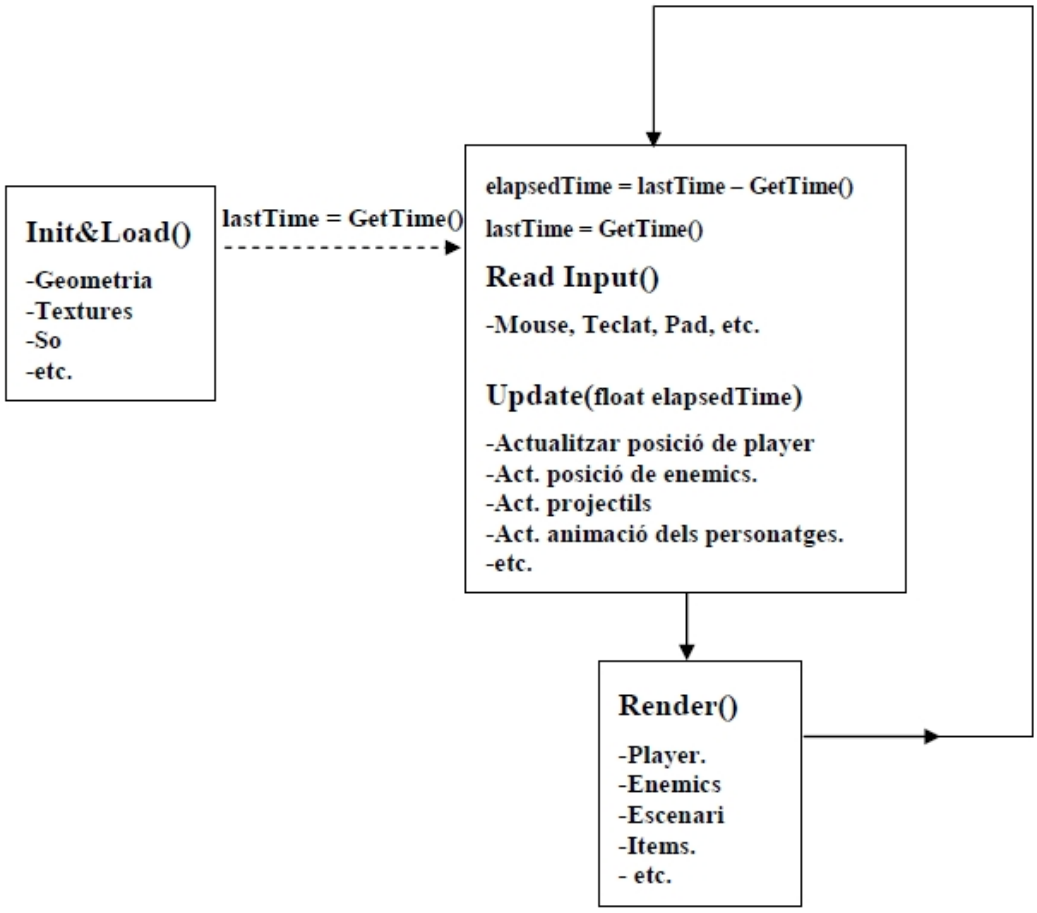


Figura 2.5: Diagrama del funcionamiento básico de Quake

Es cierto que esta estructura está ya desfasada por estar diseñada para ser ejecutada en un único hilo pero, debido a la envergadura que supone en nuestro proyecto el descubrir una tecnología compleja como la de los motores de física, no tiene sentido aventurarse en un diseño multihilo.

Otro aspecto muy importante del uso de este *framework* es la reutilización del motor gráfico, aspecto en el que hemos observado se pierde la mayoría de la gente que relata sus inicios en el foro de Havok.

Por último, destacar que no sirve de ayuda el hecho de que Quake ya incorpore una librería para el cálculo de las colisiones estáticas (ColDet[COL]), todo lo contrario, obligará a realizar la parte de rediseño más importante del proyecto.

2.3.4. Método

Para el desarrollo de este proyecto se utilizaron, de inicio, las mismas herramientas que durante el curso de videojuegos: Microsoft Visual Studio 2005, Autodesk 3ds Max 2009 y una versión más reciente del DirectX SDK (agosto 2009).

A éstas hubo que añadir el software de Havok, formado por la librería de Havok Physics, una herramienta de creación de contenidos llamada Havok Content Tools y una herramienta de visualización del mundo físico llamada Visual Debugger.

Destacar que, antes de tomar la decisión de seguir utilizando 3ds Max, exploramos toda una serie de alternativas motivados por el activo debate que existía en el foro de Havok sobre cual debía ser el mejor motor gráfico a implementar de cara a la posterior integración con Havok. Durante esta fase se experimentó con Terragen, OGRE, IRLICHT, COLLADA y Blender.

Los formatos de fichero trabajados con dichas herramientas han sido:

Microsoft Visual Studio 2005

.h, .cpp declaraciones y definiciones C++.

.xml opciones de configuración e interfaz gráfica de usuario.

Autodesk 3ds Max 2009

.max modelos 3D.

.mat biblioteca de materiales.

.x malla 3D texturizada en formato DirectX (generada mediante el *plugin* Panda DirectX Exporter[PDE]).

.hkx malla 3D formada por una o más entidades físicas (generada mediante Havok Content Tools).

2.4. Quake goes racing!

Para el desarrollo de este proyecto se siguió el método en espiral[WKp] con la intención de poder, una vez realizada la implementación básica de Havok, ir añadiendo nuevas funcionalidades a medida que fuéramos descubriendo más sobre el funcionamiento de dicho motor de física.

2.4.1. Requisitos

Para cada nueva fase de rediseño e implementación, se establecieron unos requisitos base a cumplir siempre.

Requisitos funcionales

- Havok debe simular las colisiones estáticas de un terreno 3D.
- Havok Vehicle Kit debe simular el comportamiento de un vehículo sobre un terreno 3D.
- Los modelos 3D que utilicemos deben ser mallas 3D optimizadas (en cuanto a número de polígonos) para su uso en videojuegos.

Requisitos no funcionales

- El módulo que integra mediante Havok las colisiones estáticas y el comportamiento del vehículo, debe consumir la mínima memoria y CPU posible para que el *frame rate* del videojuego no se vea afectado.
- El comportamiento del vehículo debe reaccionar de manera realista al manejo del jugador.

2.4.2. Diseño

El rediseño del *framework* de Quake fue una de las tareas más complejas que tuve que superar a lo largo de este proyecto. Ello fue debido a cómo funciona ColDet, la librería que se encargaba de las colisiones estáticas. En un principio, uno podría pensar que la migración a Havok sería tan simple como la sustitución de una librería por otra, y que luego el mérito y las horas de trabajo se irían en readaptar el código de cada una de las funciones del módulo de colisiones ya existente. Sin embargo, la realidad es que esa opción, tan siquiera existe.

Cómo decíamos, todo se debe a cómo funciona Coldet. Para que nos hagamos una idea de lo alejado que está ColDet de un motor de física, ColDet es una librería que únicamente es capaz de resolver un simple fenómeno físico: la detección de colisiones entre 2 cuerpos. Así pues, a la pregunta de si un cuerpo está chocando con otro, ColDet puede respondernos sí o no además de en qué punto, ¡pero nada más! ColDet, a diferencia de un motor de física, no simula mundo físico alguno, sino que tan sólo realiza ciertos cálculos geométricos 3D para resolver si un cuerpo poliédrico está atravesando otro. Así pues, ColDet ni simula la fuerza de la gravedad ni calcula la reacción de dos cuerpos al colisionar (lógico, no conoce el peso de ninguno de los objetos, ni las características físicas tales como fricción o elasticidad del material del objeto, o la velocidad a la que se desplaza), tan sólo nos proporciona la lógica necesaria para que nosotros simulemos mediante código el correcto comportamiento físico que debe ocurrir en pantalla.

Y aquí llega la gran dificultad que atravesamos en este punto del proyecto, comprender que Havok requiere un cambio de concepto radical: simular un mundo físico completo a parte y paralelo a nuestro mundo gráfico. Cómo ha de ser dicho mundo físico es algo que explicamos más adelante (apartado 2.5.2), pero sí es importante destacar que este cambio de enfoque tiene serias consecuencias a nivel de diseño, ya que cambia completamente el diálogo entre módulos y la secuencia de acciones a realizar. Antes el mundo gráfico llevaba la batuta, hacía preguntas al detector de colisiones en momentos puntuales tras pasarle cierta información y, en función de su respuesta, tomaba una decisión u otra. Ahora es el mundo físico quién, no sólo lleva la batuta, sino que recibe constantemente toda la información y hace la toma de decisiones, por lo que el mundo gráfico se limita a preguntarle, literalmente: "¿dónde me pinto?"

Por último, destacar que tras diferentes pruebas a la hora de decidir los formatos de las mallas 3D con las que trabajar, decidimos descartar los ficheros .ASE (para usar .X en su lugar), pero ello no implicó ningún cambio al diseño original, tan sólo reimplementar las funciones *Load()* y *Render()* de la clase ASEObject, ahora renombrada XObject.

2.4.3. Implementación

Physics Manager

Ésta es la clase que hemos desarrollado casi a lo largo de todo el curso. Integra Havok y su *vehicle kit* en Quake.

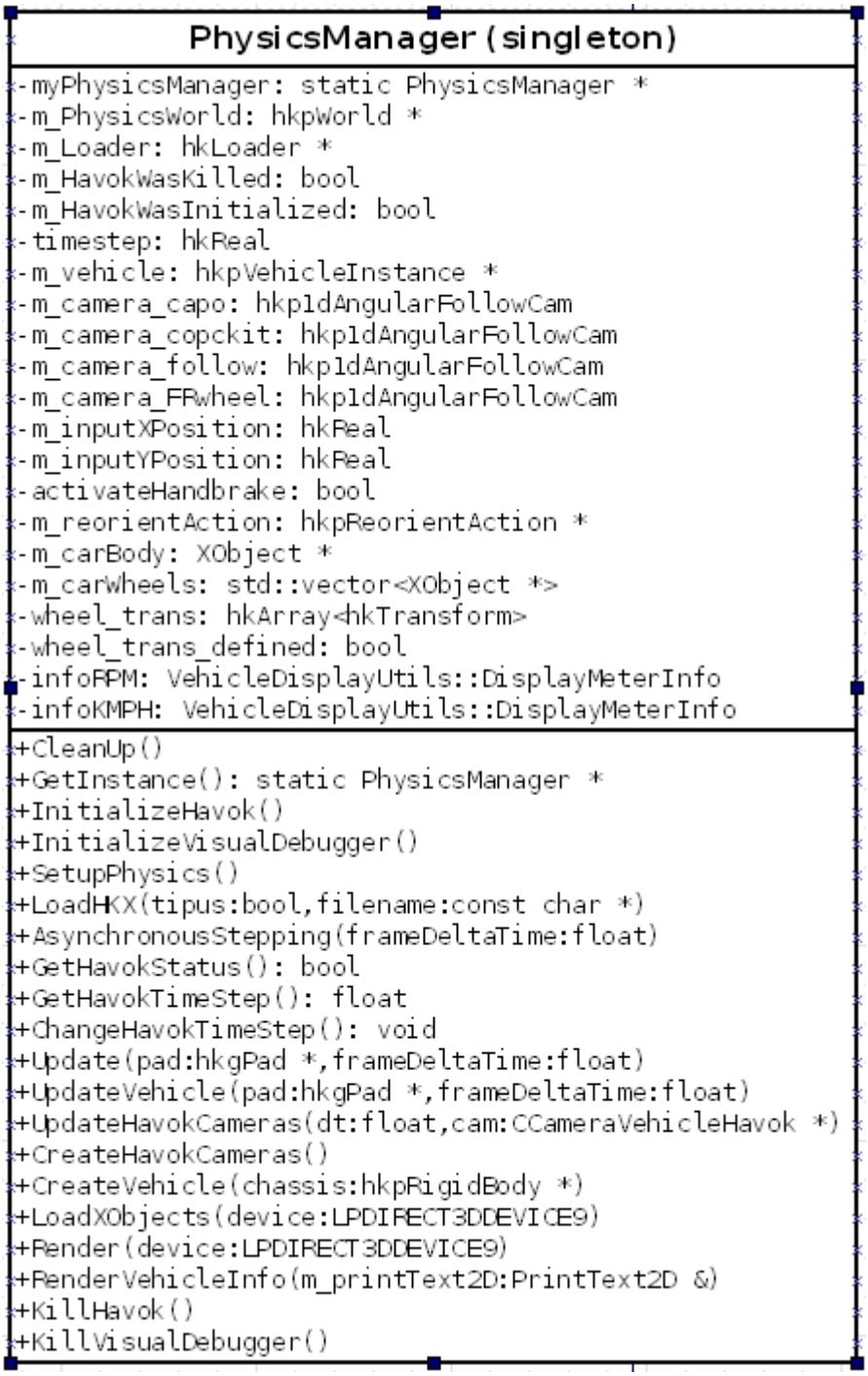


Figura 2.6: Diagrama de la clase

PhysicsManager() inicialización de variables a su valor *default*, una de ellas en especial: la frecuencia a la que funcionará Havok.

InitializeHavok() inicializa todos los sistemas base de Havok (memoria, número de hilos y cola de procesos), define los parámetros del mundo físico (tales como tipo, tamaño, modo de funcionamiento) y lo crea.

CleanUp() libera la memoria.

GetInstance() retorna la instancia de nuestro singleton.

InitializeVisualDebugger() inicializa el proceso que enviará a VisualDebugger toda la información de lo que suceda en el mundo físico.

SetupPhysics() define los parámetros del vehículo (tales como posición, peso y dirección del chasis), lo crea y lo añade al mundo físico.

LoadHKX() obtiene de un fichero .HKX toda entidad física definida y, junto con sus propiedades, la añade al mundo físico.

AsynchronousStepping() hace que el tiempo corra en el mundo físico, es decir, avanza la simulación.

GetHavokStatus() obtiene si Havok ha sido inicializado o no, es decir, si existe mundo físico alguno.

GetHavokTimeStep() obtiene a qué frecuencia está funcionando Havok, que será un valor de 30 o 60 Hz.

ChangeHavokTimeStep() función que nos permite cambiar en tiempo real a qué frecuencia trabaja Havok.

Update() lleva a cabo toda la logística que supone la actualización del mundo físico.

UpdateVehicle() actualiza la posición del vehículo en caso de que el jugador haya utilizado alguno de los mandos de control, incluida la opcion de recolocar.

UpdateHavokCameras() actualiza la vista de la cámara correspondiente, según la que use el jugador en cada momento.

LoadXObjects() carga en memoria las mallas 3D y las texturas que componen el vehículo.

Render() pinta en pantalla las mallas 3D que componen el vehículo.

RenderVehicleInfo() pinta en pantalla los velocímetros del vehículo.

2.5. Creación de contenidos

Hemos añadido este apartado a la memoria para mostrar todo el trabajo que hemos realizado en tareas, digamos, colaterales a las que deberían ser las de un ingeniero informático. Curiosamente, esa es precisamente la característica a resaltar en esta sección, ya que la realización de dichas tareas nos ha hecho comprender que el diseño de un videojuego afecta incluso hasta cómo ha de ser diseñada una malla 3D. En otras palabras, hemos descubierto cuán importante es que en el desarrollo de un videojuego todos los departamentos estén perfectamente comunicados entre ellos, cuando siempre habíamos imaginado una mayor independencia entre los mismos.

Volviendo al tema, hemos realizado el trabajo propio de un diseñador gráfico 3D, trabajando con un conjunto de modelos 3D a partir de los cuales hemos generado nuestro mundo gráfico (ficheros .X) y nuestro mundo físico (ficheros .HKX).

2.5.1. Mundo gráfico

rFactor es, además de uno de los mejores simuladores del mercado, un producto famoso por la facilidad que ofrece a la comunidad de usuarios a la hora de realizar modificaciones y/o crear nuevos contenidos. Bajo esa premisa, descargamos la demo junto con el pack de plugins para Autodesk 3ds Max 2009 que la propia web oficial ofrece y, efectivamente, un par de días después ya trabajábamos con los modelos 3D que acompañan al juego tras la inestimable ayuda de Internet.

En nuestro caso elegimos un coche de la NASCAR (famosa competición americana por las carreras que disputan en óvalos) y el circuito de Montmeló. Si bien la elección del circuito de Barcelona no tuvo otro motivo más que sentimental por el mero hecho de correr en casa, la elección del coche de la NASCAR sí tuvo un claro porqué: la forma de su carrocería era la más idónea a simular en el mundo físico por su uniformidad (sin cambios bruscos) y ello simplificaría las posibles reacciones que el vehículo sufriría en las colisiones y nos facilitaría la evaluación de si lo que ocurría en el mundo físico era correcto o no.

A medida que fuimos aprendiendo a defendernos con 3ds Max, empezamos a simplificar los modelos 3D de rFactor y empezamos las pruebas de exportación a ficheros .ASE, que era el formato con el que trabajaba Quake. Sin embargo y, tal y como nos advirtió nuestro tutor, la clase ASEManager que crearon en su día no era ninguna maravilla por lo que nos vimos obligados a buscar otras alternativas, apareciendo en escena el formato .X, propio de DirectX y al que di una oportunidad tras ver como en el foro de Havok lo citaban muy a menudo, incluidos los miembros de Havok. La verdad es que fue un gran descubrimiento ya que, a diferencia del formato .ASE, ahora podíamos configurar una gran cantidad de opciones durante la exportación, algunas de ellas tan interesantes como la corrección del sistema de coordenadas de 3ds Max al sistema de la mano izquierda o la optimización y compresión (transformando a binario) del modelo.

Estos modelos que generamos en formato .X, tras varios intentos y meteduras de pata que sufrimos especialmente por problemas con las texturas y que nos obligaron a realizar la ardua tarea de retexturizar cada uno de los modelos son, hoy en día, los componentes de nuestro mundo gráfico, es decir, representan los gráficos que se despliegan en nuestro videojuego.

2.5.2. Mundo físico

El mundo físico es, como dice la palabra, la representación física del mundo gráfico. Ahora bien, ¿qué relación hay entre ambos mundos? Entre ambos mundos hay una relación geométrica, cuyo grado de similitud puede variar desde la exactitud total hasta una aproximación poliédrica más o menos acertada.

El quid de la cuestión reside en qué forma poliédrica elijamos para la representación física de cada objeto y, como decíamos, existe la posibilidad de hacer una representación exacta, que se da cuando elegimos la malla 3D original del objeto como su propia representación física. Dada la existencia de esta posibilidad, ¿porqué entonces utilizar cualquier otro método? La respuesta es sencilla: si bien es factible utilizar la malla 3D de todo objeto como su propia representación física y gráfica, ello no es viable en términos de memoria y CPU.

Pensemos en el mundo gráfico por un momento: los modelos 3D que aparecen en pantalla son realmente de una calidad espectacular, ¿cierto? Pues la realidad es que sus mallas han sido simplificadas en miles de polígonos aunque ello pase desapercibido para el espectador. ¡Incluso las texturas sufren estas "rebajas" en decenas de megabytes! Pues bien y, retomando el tema, en el mundo físico se realiza un proceso parecido y se busca reducir al máximo el número de polígonos que representa a cada objeto, todo ello con el claro propósito de simplificar enormemente la complejidad de los cálculos que el motor de física deberá realizar. Por ello, a la opción de usar la malla original como representación física, se le suman los poliedros convexos, el cilindro, la cápsula y, como unidades más simples, la esfera y la caja.

Pero aún se puede hilar más fino y encontrar una unidad de representación física más simple: la nada. Efectivamente, no siempre es necesario reproducir todo el mundo gráfico en el mundo físico. En nuestro caso, las gradas, los pisos superiores en el edificio de boxes, los puentes que cruzan la pista con publicidad en sus pasarelas... son objetos contra los que nuestro vehículo NASCAR nunca impactará, por lo que es inútil computarlos.

Bien, algo que no he dicho es cómo asignamos a cada objeto el modelo geométrico que lo representará físicamente. Aquí es donde entra en escena Havok Content Tools, una de las fantásticas herramientas que incluye el software de Havok. Su instalación (como plugin) añade a 3ds Max la funcionalidad de dotar de a los objetos de características físicas tales como masa (kg), fricción, elasticidad y, cómo no, malla geométrica con la que ser representado. Todo ello es así previo la creación mediante su propia barra de herramientas de un cuerpo rígido.

En resumen, para la creación de nuestro mundo físico, primero seleccionamos en 3ds Max las partes del mundo gráfico a representar en él. A continuación, por cada una de ellas creamos un cuerpo rígido mediante la barra de herramientas de Havok Content Tools y asignamos, según el tipo de objeto, unas características físicas u otras según el tipo de material (una pila de neumáticos es más elástica que un muro y la hierba ofrece menos fricción que el asfalto). Por último, y también para cada objeto, seleccionaremos el modelo geométrico que mejor se adapte a su fisonomía a la vez que sea el más óptimo, intentando reservar el uso de la malla original para los casos insalvables, como el trazado del circuito en nuestro caso.

He dejado la masa para el final por tener ésta una funcionalidad especial. En un mundo físico, suele existir un terreno (que hace de suelo) en el que se sostienen los diferentes objetos a causa de la fuerza de la gravedad. Esos objetos sufren dicha fuerza en todo momento, como nosotros en la vida real (pregúntenle a su trasero sino), pero es obvio que la mayoría del tiempo su estado no cambia y no se mueven, siendo pues un gasto innecesario de cómputo. Por ello, Havok, que ya utiliza ciertos métodos para minimizar los recursos a utilizar en dichas situaciones, nos da la opción de facilitarle el trabajo indicándole qué objetos nunca se moverán y, en consecuencia, permanecerán fijos en el escenario. Ello se hace asignándole a un objeto una masa de 0.0 kg. Así pues, en nuestro caso, absolutamente todos los objetos del circuito son fijos y tan sólo al coche se le asignó una masa distinta de 0.0 kg.

Una vez finalizada la asignación de propiedades físicas, la barra de herramientas de Havok Content Tools nos permite exportar nuestro mundo físico a un fichero de formato .HKX y ahí finaliza el proceso. Ahora ya será la clase *PhysicsManager* quien se encargue de trabajar con él, extrayendo cada una de las entidades físicas junto con sus propiedades e insertándolas en el mundo físico de nuestro videojuego.

Destacar que durante la fase de exportación, el plugin posee una variada cantidad de filtros a aplicar que multiplican todavía más su funcionalidad.

Capítulo 3

Resultados

La primera parte de este capítulo luce las metas logradas durante la realización del proyecto. La segunda parte pone en evidencia las carencias del mismo.

3.1. Objetivos logrados

Si bien al inicio del proyecto nos marcamos una serie de objetivos (los enunciados en el apartado 1.2), a medida que fue transcurriendo el tiempo y gracias a que logramos ir resolviendo con éxito las diferentes problemáticas listadas en el apartado 1.3, creció en mí la intención de dar una vuelta de tuerca más al proyecto y me planteé el objetivo de lograr realizar lo que reza el título de este proyecto: un videojuego.

Un objetivo sin duda atrevido con la clara finalidad de dar forma al trabajo realizado hasta el momento, en el que influyó mucho la lectura de un interesantísimo artículo[HOW] en el que entendí cuán importante y difícil es terminar un videojuego, y que se convirtió en mi meta durante el tramo final del proyecto.

A continuación, una muestra de los resultados conseguidos en la realización de este proyecto que, creemos, cumplen con los objetivos inicialmente establecidos, los requisitos formulados y, finalmente, hacen de él un videojuego:

- Para la representación gráfica de nuestro vehículo y nuestro terreno 3D hemos sido capaces de trabajar con modelos 3D del juego rFactor, uno de los mejores simuladores del mercado en la actualidad. Un resultado muy ambicioso dada la complejidad en cuanto a número de polígonos de dichas mallas 3D. Para el vehículo se ha utilizado un coche de la NASCAR (figura 3.1), y como terreno el circuito de Montmeló (figura 3.2).

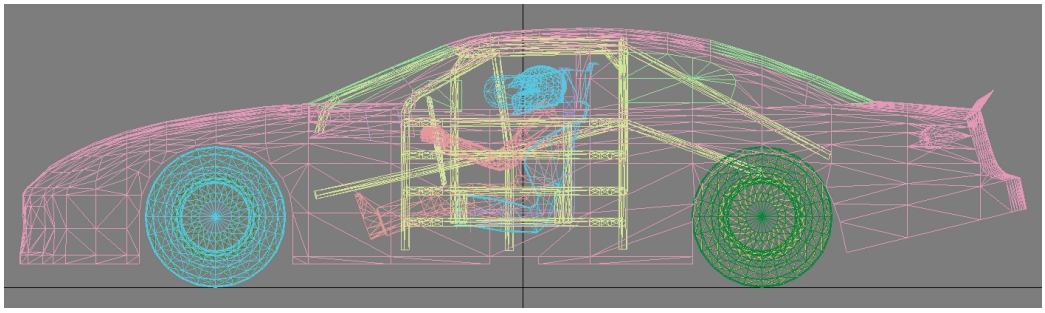
- Para la simulación física de las colisiones estáticas hemos utilizado Havok, creando una representación geométrica adecuada de nuestro vehículo y nuestro terreno para el mundo físico y, lo más importante, sincronizando éste con el mundo gráfico a escala métrica y temporal.

- Para la simulación física del comportamiento de un vehículo hemos utilizado el *vehicle kit* de Havok. El resultado ha sido fantástico debido a que la respuesta del vehículo a los diferentes cambios introducidos en sus parámetros de configuración ha sido muy positiva: por ejemplo, aumentar el coeficiente de fricción de los neumáticos aumenta el agarre en curva, pero disminuye la velocidad punta en recta, y endurecer las suspensiones disminuye el balanceo en curva y en frenada, pero entonces tiende a subvirar en la entrada de las curvas.

- La integración de la librería Havok en Quake no ha afectado al *frame rate* del videojuego, a pesar del uso de gráficos de mayor peso. Un resultado considerable, especialmente si tenemos en cuenta que cambiar entre las frecuencias de funcionamiento de Havok (30 y 60 Hz) tan sólo penaliza entre una y dos imágenes por segundo.

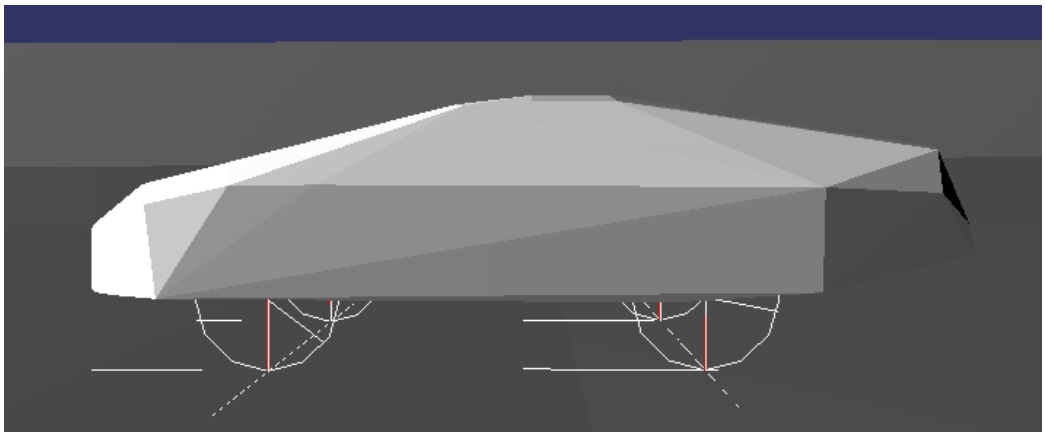
- Se ha modificado la interfaz gráfica de usuario de Quake, tanto el menú principal como el modo de juego (figura 3.3), siendo en este último donde las nuevas funcionalidades que se han añadido dotan de una finalidad concreta al videojuego: poder entrenar en un circuito y probar de establecer en él el mejor tiempo posible. Para una explicación detallada de la nueva interfaz de usuario y sus nuevas funcionalidades, consulte el manual de usuario en el anexo A.

Ya para finalizar, otro gran objetivo logrado ha sido el uso y conocimiento que he adquirido de herramientas que para mí eran absolutas desconocidas antes de la realización de este proyecto: Terragen, Autodesk 3ds Max 2009 y, muy especialmente, LaTeX, con el que he realizado esta memoria.



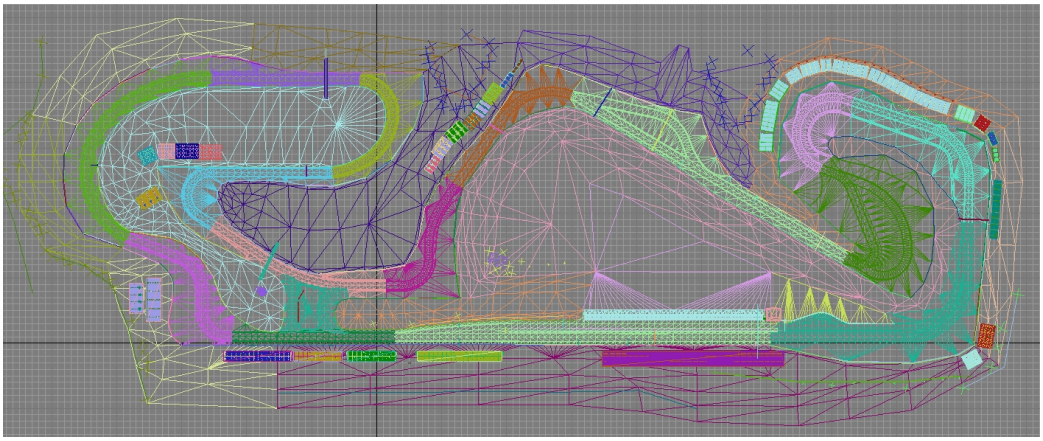
(a) Malla 3D

(b) Mundo gráfico

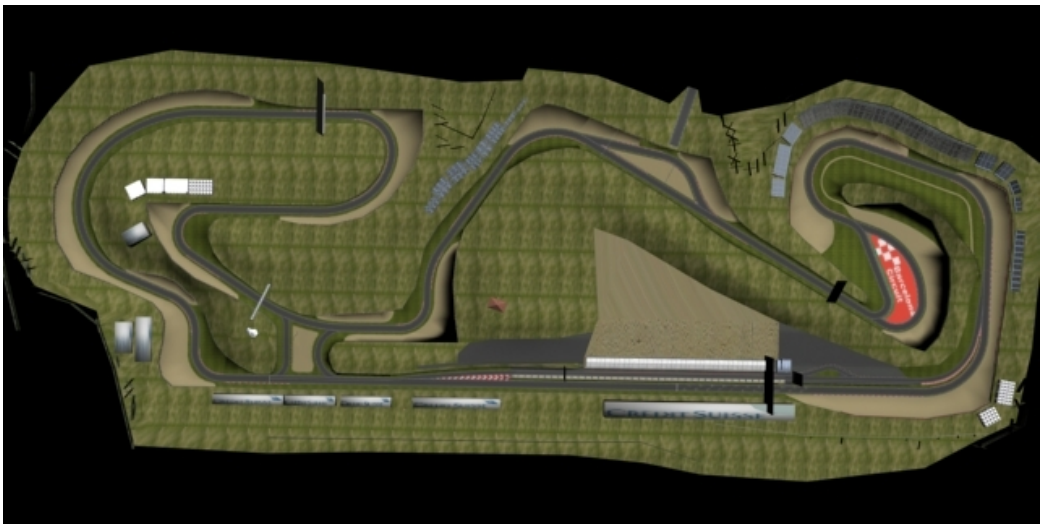


(c) Mundo físico

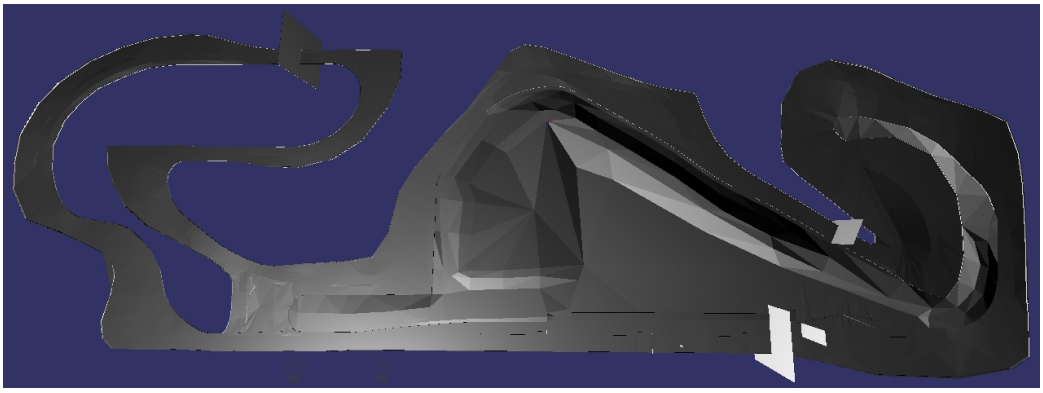
Figura 3.1: El vehículo: un coche de la NASCAR



(a) Malla 3D

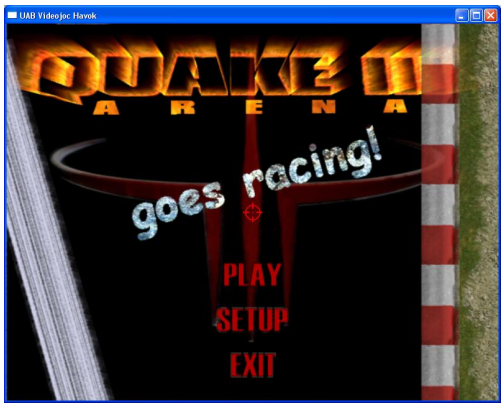


(b) Mundo gráfico



(c) Mundo físico

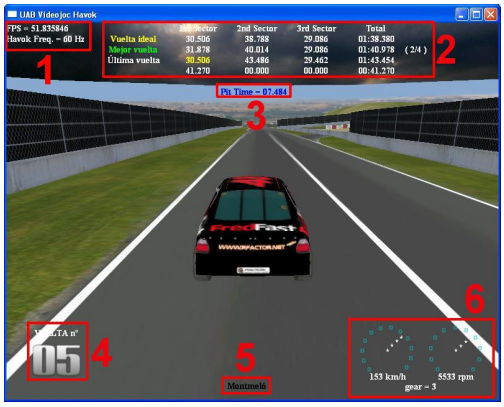
Figura 3.2: El terreno: el circuito de Montmeló



(a) Menú principal

(b) *Play*

(c) *Setup*



(d) Modo de juego

Figura 3.3: La interfaz gráfica de usuario del juego

3.2. Objetivos no logrados y fallos conocidos

No todo ha sido un camino de rosas y también se han cometido fallos (algunos más graves que otros) o no siempre se ha logrado el resultado deseado en algunas áreas del proyecto.

Siendo crítico, considero que el mayor error lo realicé en el estudio de viabilidad del proyecto (descrito en el apartado 1.7) al no considerar Autodesk 3ds Max entre los requisitos software necesarios. Estimé que en caso de necesitar alguna herramienta de diseño 3D (además de Terragen), la solución de código abierto Blender cubriría mis necesidades. Sin embargo y, una vez bastante avanzado el proyecto, descubrí que Havok Content Tools sólo trabajaba con 3ds Max, Maya y XSI.

Es obvio que el completo desconocimiento del mundo en el que me metí a la hora de realizar este proyecto no ayudó pero, personalmente y de cara al futuro, me ha quedado claro que en el estudio previo de un proyecto no se puede dejar cabo suelto alguno y que las estimaciones hay que delimitarlas al máximo. Para cerrar el asunto, destacar que me salvó el hecho de que pude sobrevivir con versiones oficiales de prueba de 3ds Max 2009.

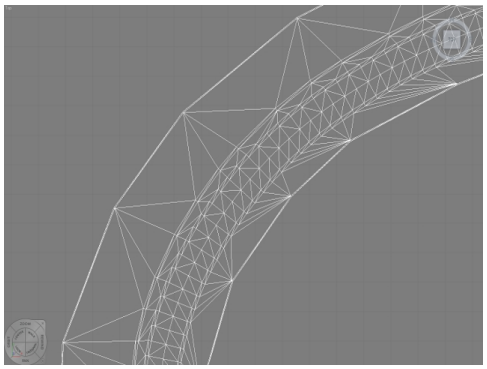
En cuanto al resto de errores que mencionaré a continuación, decir que son fallos pertenecientes a la fase de implementación (funcionalidades que se quedaron a medio camino o que no lograron implementarse correctamente del todo) que fueron identificados durante las continuas pruebas que fuimos realizando al proyecto. Sin embargo, queremos reconocer abiertamente que no hemos realizado la cantidad de tests necesarios por una sencilla razón: ¡no se puede!

La realización de este proyecto me ha hecho comprender que la dimensión de un videojuego es de tal magnitud que requiere la existencia de un equipo dedicado única y exclusivamente a la parte de tests.

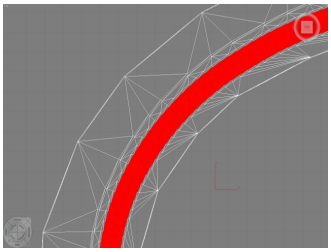
Ya sin más dilación procedo a listar los errores:

- En el circuito, las zonas de hierba y grava poseen el mismo agarre que las zonas asfaltadas. Ello se debe a una limitación de cómo funciona Havok Content Tools y el cómo fueron diseñados los modelos 3D que importé de rFactor, ya que si bien el modelo 3D sí es capaz de distinguir cada una de esas zonas para su correcto texturizado tal y como se puede observar en la figura 3.4, todas pertenecen al mismo objeto 3D, cuando Havok Content Tools tan sólo es capaz de crear cuerpos rígidos de objetos 3D al completo y no de alguna de sus partes. Ello implica asignar a todo el objeto el mismo coeficiente de fricción, siendo pues indiferente asfalto de hierba y grava.

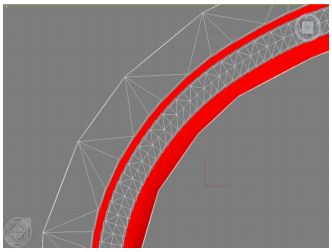
(a) Vista aérea en el juego



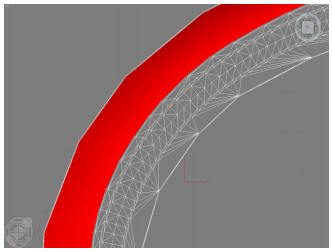
(b) Malla 3D



(c) Asfalto



(d) Hierba



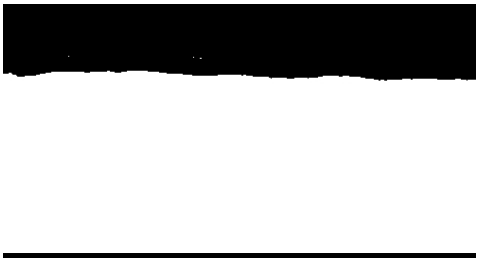
(e) Grava

Figura 3.4: Detalle del diseño gráfico de la curva Renault

- No todas las texturas del coche y el circuito conservan correctamente su canal alfa, que es el que contiene el nivel de transparencia. Ello se debe a ciertos problemas durante la importación y exportación de los modelos de rFactor en 3ds Max, por lo que es un problema del método utilizado, ajeno a nuestro código. Puede verse un ejemplo en la figura 3.5, donde se observa como el cielo nublado y el horizonte no casan.

(a) Vista en el juego

(b) Textura original



(c) Canal alfa en negro

Figura 3.5: Detalle de la problemática con las texturas alfa

- El tiempo por vuelta de nuestro cronómetro no se detiene cuando en el juego activamos la pausa. Ello es debido a que la clase *CTimer* que incorporaba Quake trabaja contra la función de Microsoft Windows *timeGetTime()*[1], resultando la estructura existente imposible de alterar para nuestro propósito.

- Nuestro código contiene *memory leaks*[2]. Desafortunadamente no se corrigió a tiempo por haber siempre otras tareas pendientes de mayor prioridad. Destacar que Quake también los tenía de origen, lo que no ayudó en absoluto.

[1]Función que retorna el tiempo (en milisegundos) que ha transcurrido desde que se inició el sistema Windows[MSC].

[2]Error de software que ocurre cuando un bloque de memoria reservada no es liberada en un programa de computación[WKq].

Capítulo 4

Conclusiones y posibles mejoras

La primera parte del capítulo presenta al lector una valoración global de la realización del proyecto que poco a poco indaga en lo que el autor considera han sido algunos de los elementos clave del mismo. La segunda parte del capítulo reflexiona sobre las líneas futuras de desarrollo a seguir y/o a completar en vistas a que se le quisiera dar continuidad al proyecto.

4.1. Conclusiones

Nueve son los meses de trabajo que nos ha costado llegar hasta aquí y, cuando echamos la vista atrás y vemos lo duro que ha sido recorrer el camino, no podemos sino alegrarnos y estar orgullosos de todo lo aprendido y de lo que finalmente se ha logrado. Sin embargo, siempre hay margen de mejora y, para ello, nada mejor que poner en marcha el sentido de la autocrítica.

¿Fue correcta la elección del motor de física? Esa es, para nosotros, la gran pregunta entorno a la cual gira todo lo demás (lógico, habiendo sido Havok la piedra angular de nuestro proyecto), y la razón principal para dudar de dicha elección es la enorme cantidad de horas que han sido necesarias para lograr sacar adelante el proyecto.

Sin embargo, nuestra respuesta (sobre si la elección del motor fue acertada o no) es un sí rotundo. Si bien Havok resultó ser bastante más complejo de lo inicialmente previsto, nuestra mayor problemática no residió en Havok en sí, sino en un tema de concepto, en el cambio de mentalidad que esta nueva tecnología requiere para la correcta comprensión de su funcionamiento: la necesidad de la existencia de dos mundos entrelazados (físico y gráfico) y cómo lograr la correcta sincronización de ambos.

Una vez superada esa fase, trabajar con Havok ha sido duro, sí, pero también muy gratificante. La extensa documentación, demos de ejemplo y el soporte oficial en el foro nos han permitido ir avanzando poco a poco y resolver los problemas que fueron apareciendo tanto con el vehículo como con el terreno 3D. Es más, sin tal cantidad de recursos, nos hubiera sido absolutamente inviable ya no finalizar el proyecto, sino realizarlo. Por ello, nos atrevemos a afirmar que la elección de Havok ha sido uno de los éxitos de este proyecto, pues con otro motor seguro habríamos tirado la toalla.

Una vez ya resuelta la duda principal, la ya mencionada elevada complejidad de Havok (que, como hemos sabido por otros miembros del departamento, es extrapolable al resto de motores completos de física) nos hace plantearnos cuál será la evolución de esta tecnología en el futuro. Si bien las últimas actualizaciones (publicadas durante la realización de este proyecto) van en la línea de lograr una mayor abstracción en pro del desarrollador final, los avances son reducidos. Por ello, se están empezando a dibujar nuevas alternativas en el mercado que nos han parecido muy interesantes:

Physics Abstraction Layer (PAL) Un *framework* que aglutina la mayoría de librerías de los motores de física existentes y que, mediante una única interfaz, nos permite construir un mundo físico usando uno o ¡más motores a la vez[PAL]! Todo un logro en cuanto a nivel de abstracción.

Simul-X Un extra para Havok que abstrae la parte del *vehicle kit* y permite la creación de todo tipo de vehículos[SIM]. Lástima que saliera oficialmente a la luz en plena fase de desarrollo de este proyecto.

4.2. Posibles mejoras y/o ampliaciones

- Rediseño más modular de la clase que controla la física (*PhysicsManager*) con el fin de establecer una clase independiente del motor de física a implementar. El objetivo de esta mejora sería facilitar la implementación de cualquier otro motor de física.

- En la misma línea de antes, realizar una mejor abstracción de la clase *PhysicsManager*, ya que ésta se encarga de la carga en memoria y del renderizado de la malla gráfica del vehículo.

- Ser capaces de guardar o cargar, en cualquier instante de tiempo, el estado del mundo físico. El objetivo de esta mejora es mucho más que el hecho de permitir guardar o cargar partidas, pues sería el primer paso de cara a poder trabajar más adelante con repeticiones y con vehículos fantasma[1].

- Para la renderización del escenario se ha reutilizado el algoritmo basado en A* existente en Quake. Si bien ello nos ha dado un resultado aceptable, nuestro tipo de terreno 3D (un espacio abierto) aconseja la implementación del algoritmo de oclusión, con el que en principio se obtendrá un *frame rate* superior además de simplificar toda la parte que se encarga del renderizado del escenario.

[1] Un vehículo fantasma es aquél cuyas colisiones están desactivadas, y suele dársele la funcionalidad de mostrar al jugador la trazada de su mejor vuelta mientras éste, a su vez, completa su vuelta actual.

Referencias

[AMD] *AMD and Havok to Optimize Physics for Gaming* [en línea]. AMD, 12 de junio de 2008. Disponible en Web: http://www.amd.com/us/press-releases/Pages/-Press_Release_126548.aspx

[BUL] Varios autores. *Games, Demos or Movies Using Bullet* [en línea]. Physics Simulation Forum, agosto de 2006 [actualizado: 6 de junio de 2010]. Disponible en Web: http://www.bulletphysics.org/Bullet/phpBB3/-viewforum.php?f=17

[CAa] MARTI, Enric. *Gráficos por Computador II* [en línea]. Disponible en Web: http://caronte.uab.cat/course/view.php?id=45

[CAb] MARTI, Enric; ARNAL, Jordi; VERGARA, Enric. *Programación Básica de Videojuegos 3D* [en línea]. Disponible en Web: http://caronte.uab.cat/course/view.php?id=30

[CAc] MARTI, Enric; ARNAL, Jordi; VERGARA, Enric. *Programación Avanzada de Videojuegos 3D* [en línea]. Disponible en Web: http://caronte.uab.cat/course/view.php?id=59

[COL] Varios autores. *ColDet 3D Collision Detection* [en línea]. SourceForgel.net, 10 de noviembre de 2000 [actualizado: 5 de marzo de 2007] Disponible en Web: http://sourceforge.net/projects/coldet/

[DTX] Varios autores. *The DirectX Software Development Kit*. Microsoft Corporation, agosto de 2009. Disponible en formato CHM, CHI y HXS.

[EMG] VAN VERTH, James M; BISHOP, Lars M; CLARK, Jason; SINGER, Marq; NORDENSTAM, Marcus. *Essential Math for Games Programmers* [en línea]. essentialmath.com, 30 de abril de

2004 [actualizado: 6 de agosto de 2009]. Disponible en Web: http://www.essentialmath.com/tutorial.htm

[HAa] *Havok - Available Games* [en línea]. Disponible en Web: http://www.havok.com/index.php?page=available-games

[HAb] Varios autores. *Havok Physics/Animation Quickstart Guide*. Telekinesys Research Ltd., 2009. Disponible en formato PDF y CHM.

[HAc] Varios autores. *Havok Physics/Animation User Guide*. Telekinesys Research Ltd., 2009. Disponible en formato PDF, CHM y CHI.

[HAd] Varios autores. *Havok Content Tools User Guide*. Telekinesys Research Ltd., 2009. Disponible en formato CHM.

[HAe] Varios autores. *Havok - Intel Software Network* [en línea]. Intel, 31 de mayo de 2008 [actualizado: 10 de junio de 2010]. Disponible en Web: http://software.intel.com/en-us/forums/havok/

[HOW] HOWLAND, Geoff. *How do I make games? A Path to Game Development* [en línea]. GameDev.net, 5 de enero de 2000. Disponible en Web: http://www.gamedev.net/reference/design/features/makegames/

[INT] *Intel To Acquire Havok* [en línea]. Intel Corporation, 15 de setiembre de 2007. Disponible en Web: http://www.intel.com/pressroom/archive/releases/2007/-20070914corp.htm

[KOW] KOWALISKI, Cyril. *GeForce 8 graphics processors to gain PhysX support* [en línea]. The Tech Report, 14 de febrero de 2008. Disponible en Web: http://techreport.com/discussions.x/14147

[LJa] HECKER, Chris; LANDER, Jeff. *Product Review of Physics Engines, Part One: The Stress Tests* [en línea]. Gamasutra, 13 de setiembre de 2000. Disponible en Web: http://www.gamecareerguide.com/features/20000913/-lander_01.htm

[LJb] HECKER, Chris; LANDER, Jeff. *Product Review of Physics Engines, Part Two: The Rest of the Story.* [en línea]. Gamasutra, 20 de setiembre de 2000. Disponible en Web: http://www.gamecareerguide.com/features/20000920/-lander_01.htm

[MAR] MARCUS, Fred. *What Designers Need to Know About Physics* [en línea]. Gamasutra, 21 de enero de 2003. Disponible en Web: http://www.gamasutra.com/resource_guide/20030121/-marcus_01.shtml

[MGC] Varios autores. *David Kaemmer, Developer BIO* [en línea]. MobyGames. Disponible en Web: http://www.mobygames.com/developer/sheet/view/developerId,4/

[MSC] Varios autores. *timeGetTime Function* [en línea]. Microsoft Corporation, 6 de abril de 2010. Disponible en Web: http://msdn.microsoft.com/en-us/library/dd757629%28VS.85%29.aspx

[NVI] HARA, Michael; PEREZ, Derek. *NVIDIA to Acquire AGEIA Technologies* [en línea]. Nvidia, 4 de febrero de 2008. Disponible en Web: http://www.nvidia.com/object/io_1202161567170.html

[NOT] NOTLAHTOLLI. *Evaluating Physics Engines For Games - PAL -* [en línea]. Experiencies in 3D, 7 de diciembre de 2008. Disponible en Web: http://3dexperiences.blogspot.com/2008/12/evaluating-physics-engines-for-games.html

[OSI] *Open Source Initiative OSI - The zlib/libpng License* [en línea]. Disponible en web: http://www.opensource.org/licenses/zlib-license.php

[PAL] BOEING, Adrian. *PAL: Physics Abstraction Layer* [en línea]. AdrianBoeing.com, 15 de febrero de 2009. Disponible en Web: http://www.adrianboeing.com/pal/index.html

[PDE] TATHER, Andy. *Panda DirectX Exporter* [en línea]. Pandasoft, 12 de agosto de 2004 [actualizado: 11 de abril de 2010]. Disponible en Web: http://www.andytather.co.uk/Panda/directxmax.aspx

[PHX] *Featured PhysX Titles* [en línea]. Nvidia Developer Zone [actualizado: 13 de febrero de 2009]. Disponible en Web: http://developer.nvidia.com/object/physx_good_company.html

[PHT] Varios autores. *Physics Tutorials* [en línea]. GameDev.net, 31 de julio de 1999 [actualizado: 15 de diciembre de 2007]. Disponible en Web: http://www.gamedev.net/reference/list.asp?categoryid=139

[RHO] RHODES, Graham. *List of physics engines and reference material* [en línea]. GameDev.net, 12 de diciembre de 2007 [actualizado: 23 de febrero de 2010]. Disponible en Web: http://www.gamedev.net/community/forums/-topic.asp?topic_id=475753

[SIM] Varios autores. *Simul-X.com* [en línea]. Simul-X.com, 15 de junio de 2009 [actualizado: 7 de febrero de 2010]. Disponible en Web: http://www.simul-x.com/

[WKa] Varios autores. *Papyrus Design Group* [en línea]. Wikipedia, 3 de febrero de 2005 [actualizado: 27 de mayo de 2010]. Disponible en Web: http://en.wikipedia.org/wiki/Papyrus_Design_Group

[WKb] Varios autores. *iRacing.com* [en línea]. Wikipedia, 13 de junio de 2008 [actualizado: 2 de junio de 2010]. Disponible en Web: http://en.wikipedia.org/wiki/IRacing

[WKc] Varios autores. *Sim racing* [en línea]. Wikipedia, 6 de julio de 2004 [actualizado: 23 de mayo de 2010]. Disponible en Web: http://en.wikipedia.org/wiki/Sim_racing

[WKd] Varios autores. *Framework* [en línea]. Wikipedia, 7 de mayo de 2005 [actualizado: 18 de abril de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Framework

[WKe] Varios autores. *Arcade* [en línea]. Wikipedia, 5 de setiembre de 2004 [actualizado: 25 de mayo de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Arcade#Arcade_como_género_de_videojuegos

[WKf] Varios autores. *Jugabilidad* [en línea]. Wikipedia, 6 de mayo de 2006 [actualizado: 13 de mayo de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Jugabilidad

[WKg] Varios autores. *Havok (software)* [en línea]. Wikipedia, 26 de julio de 2004 [actualizado: 3 de mayo de 2010]. Disponible en Web: http://en.wikipedia.org/wiki/Havok_Physics

[WKh] Varios autores. *AGEIA* [en línea]. Wikipedia, 22 de agosto de 2006 [actualizado: 3 de enero de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/AGEIA

[WKi] Varios autores. *Mecánica del sólido rígido* [en línea]. Wikipedia, 22 de abril de 2006 [actualizado: 6 de mayo de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Mecánica_del_sólido_rígido

[WKj] Varios autores. *Mecánica de sólidos deformables* [en línea]. Wikipedia, 28 de diciembre de 2005 [actualizado: 11 de mayo de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Mecánica_de_sólidos_deformables

[WKk] Varios autores. *Mecánica de fluidos* [en línea]. Wikipedia, 27 de octubre de 2002 [actualizado: 6 de junio de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Mecánica_de_fluidos

[WKl] Varios autores. *Imágenes por segundo* [en línea]. Wikipedia, 2 de febrero de 2009 [actualizado: 6 de junio de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Imágenes_por_segundo

[WKm] Varios autores. *Mecánica newtoniana* [en línea]. Wikipedia, 22 de mayo de 2006 [actualizado: 9 de junio de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Mecánica_newtoniana

[WKn] Varios autores. *Renderización* [en línea]. Wikipedia, 12 de febrero de 2002 [actualizado: 8 de mayo de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Renderización

[WKñ] Varios autores. *Quake III Arena* [en línea]. Wikipedia, 22 de octubre de 2004 [actualizado: 11 de abril de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Quake_III_Arena

[WKo] Varios autores. *Videojuego de disparos en primera persona* [en línea]. Wikipedia, 8 de enero de 2003 [actualizado: 7 de junio de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/First-person_shooter

[WKp] Varios autores. *Desarrollo en espiral* [en línea]. Wikipedia, 5 de julio de 2006 [actualizado: 26 de mayo de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Desarrollo_en_espiral

[WKq] Varios autores. *Fugas de memoria* [en línea]. Wikipedia, 30 de setiembre de 2003 [actualizado: 15 de abril de 2010]. Disponible en Web: http://es.wikipedia.org/wiki/Fuga_de_memoria

Apéndice A

Manual de usuario

Este anexo describe la interfaz gráfica del videojuego *Quake goes racing!*.

A.1. Menú principal

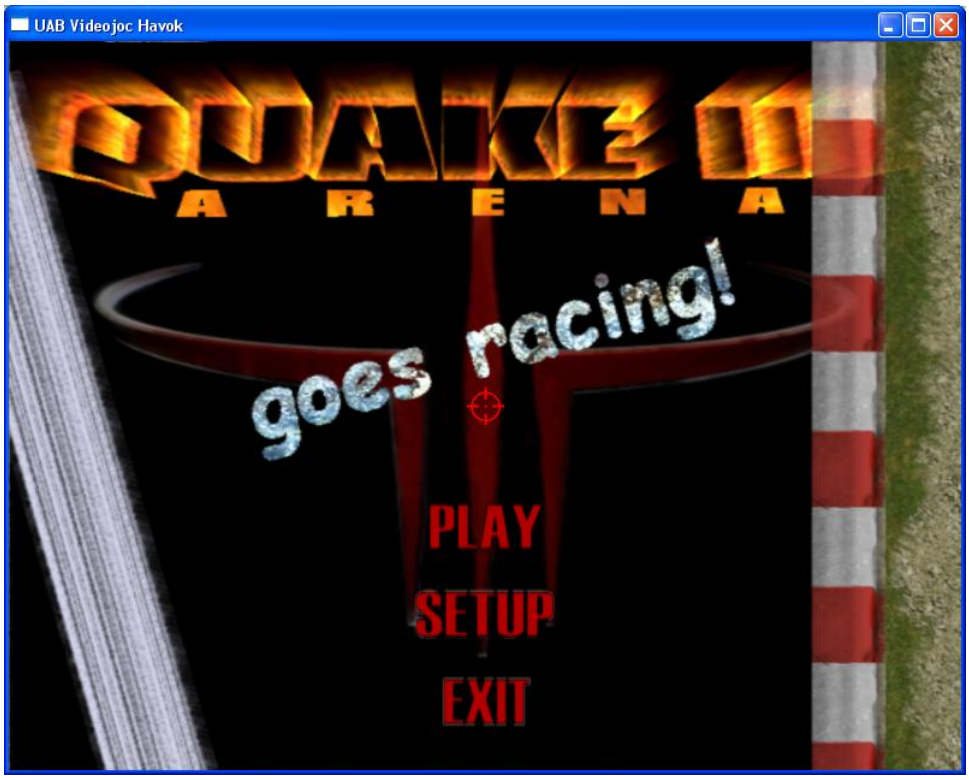


Figura A.1: Vista del menú principal

Al arrancar el juego aparece un menú (figura A.1) que nos ofrece tres opciones:

- **Play:** para iniciar una partida nueva.
- **Setup:** para configurar las opciones de rendimiento del sistema.
- **Exit:** para salir del juego.

A.1.1. Menú play

Al apretar el botón de *Play* aparece otro menú (figura A.2) que nos muestra el circuito donde se va a correr. El botón *Next* nos lleva directamente a la pista y, mientras se carga el circuito, se puede oír de fondo un ambiente de carreras que ameniza la espera. Para volver al menú principal hay que pulsar el botón *Back*.

Figura A.2: Vista del menú play

A.1.2. Menú setup

Este menú (figura A.3) nos permite configurar el nivel de detalle gráfico del videojuego:

- **Pista:** nivel mínimo de detalle, sólo aparece la pista.
- **Gradas:** nivel medio de detalle, se añaden las gradas.
- **Completo:** nivel máximo de detalle, se añaden gradas, pancartas de publicidad y público.

También puede configurarse el nivel de volumen deseado en la opcion de *Music Volume* que aparece en la parte inferior de la pantalla.

Figura A.3: Vista del menú setup

Una vez configurado el nivel de detalle y el volumen deseados, volvemos al menú principal pulsando el botón *Back*.

A.2. Interfaz en modo de juego

Cuando se inicia la partida, el coche se sitúa en la línea de meta y nos aparecen toda una serie de indicadores alrededor del vehículo (figura A.4).

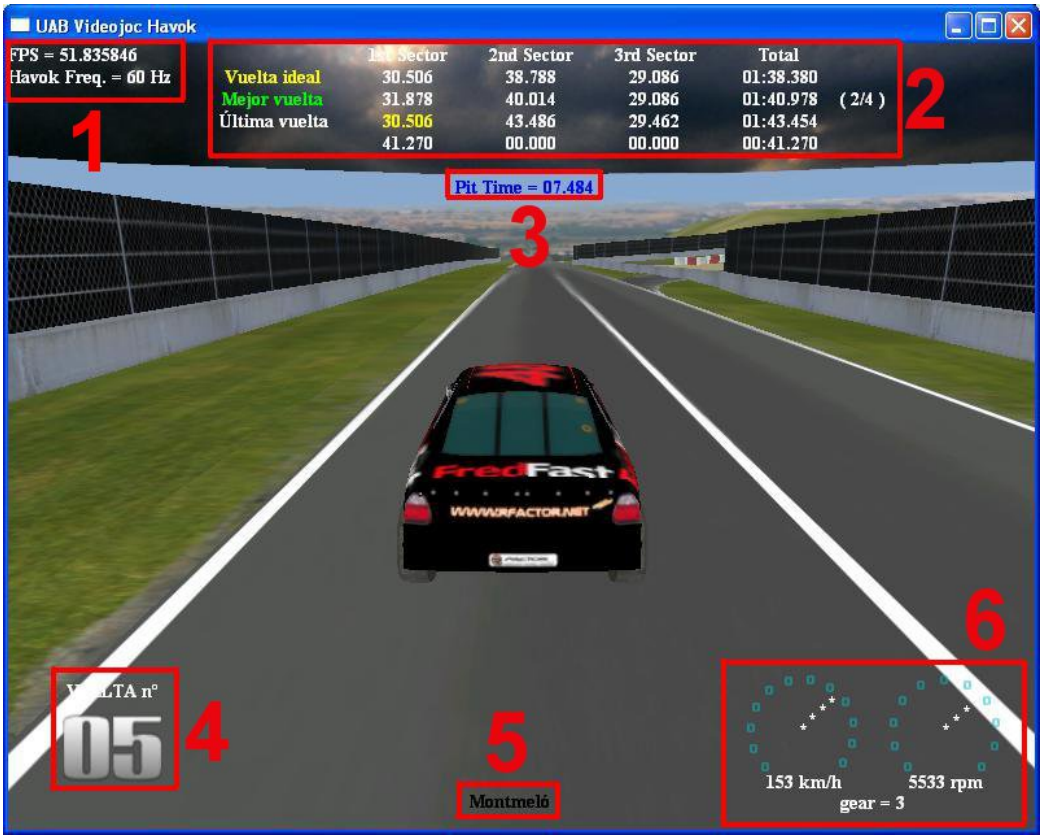


Figura A.4: Interfaz en modo de juego

1. **Rendimiento**

   En la parte superior izquierda de la pantalla aparecen los siguientes valores de rendimiento del juego:

   - **FPS:** muestra las imágenes por segundo a las que está funcionando el juego.
   - **Havok Freq.:** indica la frecuencia de trabajo del motor de física Havok.

2. **Tiempos**

   En la parte superior central de la pantalla aparecen los siguientes tiempos divididos en sectores:

   - **Vuelta ideal:** vuelta compuesta por el mejor tiempo conseguido en cada uno de los sectores. Este tiempo sirve de referencia al piloto para conocer su margen de mejora.
   - **Mejor vuelta:** tiempo más rápido conseguido hasta el momento (indica también en qué número de vuelta se registró).
   - **Última vuelta:** tiempo de la vuelta anterior.
   - **Vuelta actual:** en la última fila aparecen los tiempos de la vuelta que estamos completando.

   Los tiempos de última vuelta y vuelta actual pueden ser de tres colores diferentes: blanco, verde o amarillo. Si registras el mejor tiempo absoluto en un sector, éste se colorea de amarillo. Si mejoras el tiempo parcial de tu mejor vuelta, entonces se colorea de verde. En caso de no haber mejora alguna, el tiempo del sector se colorea de blanco.

3. **Pit time**

   En la parte central de la pantalla hay un cartel que nos indica el tiempo transcurrido durante nuestro último paso por boxes.

4. **Número de vueltas**

   En la parte inferior izquierda de la pantalla se nos indica el número de vueltas completadas.

5. **Nombre del circuito**

   En la parte inferior central de la pantalla se nos muestra el nombre del circuito en el que estamos corriendo (Montmeló en nuestro caso).

6. **Velocímetros**

   En la parte inferior derecha de la pantalla se muestran unos indicadores que nos ofrecen la siguiente información: velocidad (en km/h), revoluciones del motor (en rpm) y marcha actual (cartel *gear*).

Para acceder al menú de opciones dentro de una partida, presionamos la tecla de Escape y nos aparece un menú (figura A.5) que nos permite continuar la partida, reiniciarla o abandonarla.

Figura A.5: Menú de opciones en modo de juego

A.2.1. Modos de vista

Hay disponibles cinco cámaras diferentes a disposición del jugador:

- **Espectador:** ajena al vehículo, permite desplazarse por todo el circuito "volando".
- **Cockpit:** situada en el interior del vehículo.
- **3ª persona:** situada fuera del vehículo, ofrece una perspectiva del mismo en tercera persona y es la cámara por defecto cuando arranca la partida.
- **Capó:** situada en el capó del vehículo.
- **Rueda:** situada en la rueda delantera derecha del vehículo, muestra el trabajo conjunto de la rueda y la suspensión.

A.3. Mapa de caracteres del juego

Se puede interactuar con el juego mediante el uso de las siguientes teclas:

Tecla	Función
A	Acelerar
Z	Frenar
, (coma)	Girar izquierda
. (punto)	Girar derecha
Espacio	Freno de mano
R	Reset del coche
T	Mostrar/ocultar tiempos
I	Mostrar/ocultar velocímetros
Escape	Menú de juego
Teclas cursor*	Mover cámara “espectador”
Shift derecho	Aumentar velocidad de la cámara “espectador”
C	Elección cámara de control
V	Elección cámara de punto de vista
F	Elección cámara de *frustrum*
F7	Mostrar/ocultar detalles rendimiento
F8	Modificar frecuencia motor Havok (30/60 Hz)

Tabla A.1: Mapa de caracteres del juego

* Esta cámara también se maneja con el movimiento del ratón.

Resumen

En la carrera del mundo de los videojuegos por alcanzar insospechables cotas de realismo con las que seguir sorprendiendo y enganchando al público, los motores de física se han convertido en la herramienta de presente y futuro.

Atraídos por el auge de esta nueva tecnología, hemos lidiado con los motores referencia hoy día en el mercado, seleccionando luego uno de ellos e implementando un humilde videojuego de carreras como muestra de su potencial y de los conocimientos adquiridos.

Resum

A la cursa del món dels videojocs per arribar a límits insospitables de realisme amb els que continuar sorprenent i atrapant al públic, els motors de física han esdevingut l'eina de present i futur.

Atrets per l'apogeu d'aquesta nova tecnologia, hem treballat amb els motors referència avui dia al mercat, seleccionant-ne un d'ells i implementant un humil videojoc de curses com a mostra del seu potencial i dels coneixements adquirits.

Resume

In the never ending battle to achieve the highest realism level, video games have found in physics engines the key for the present and the future.

Because of the recent boom of this new technology, we looked at some of the most known engines to finally select one and implement a humble racing game to demonstrate its potential and our knowledge.